



TITLE:

An exact algorithm for the precedence-constrained single-machine scheduling problem

AUTHOR(S):

Tanaka, Shunji; Sato, Shun

CITATION:

Tanaka, Shunji ...[et al]. An exact algorithm for the precedence-constrained single-machine scheduling problem. European Journal of Operational Research 2013, 229(2): 345-352

ISSUE DATE:

2013-09

URL:

<http://hdl.handle.net/2433/174100>

RIGHT:

© 2013 Elsevier B.V.; This is not the published version. Please cite only the published version.; この論文は出版社版ではありません。引用の際には出版社版をご確認ご利用ください。

An exact algorithm for the precedence-constrained single-machine scheduling problem

Shunji Tanaka^{a,*}, Shun Sato^a

^a*Department of Electrical Engineering, Kyoto University
Kyotodaigaku-Katsura, Nishikyo-ku, Kyoto 615-8510, Japan*

Abstract

This study proposes an efficient exact algorithm for the precedence-constrained single-machine scheduling problem to minimize total job completion cost where machine idle time is forbidden. The proposed algorithm is based on the SSDP (Successive Sublimation Dynamic Programming) method and is an extension of the authors' previous algorithms for the problem without precedence constraints. In this method, a lower bound is computed by solving a Lagrangian relaxation of the original problem via dynamic programming and then it is improved successively by adding constraints to the relaxation until the gap between the lower and upper bounds vanishes. Numerical experiments will show that the algorithm can solve all instances with up to 50 jobs of the precedence-constrained total weighted tardiness and total weighted earliness-tardiness problems, and most instances with 100 jobs of the former problem.

Keywords:

scheduling, single-machine, precedence constraints, exact algorithm, Lagrangian relaxation, dynamic programming

1. Introduction

In this study we consider the single-machine scheduling problem to minimize total job completion cost subject to general precedence constraints ($1|prec|\sum f_j(t)$). It is often the case in practice that some job should be processed before another job due to limitation of machine functions, tool change restrictions and so on. This type of constraint is called precedence constraint.

*Tel.: +81-75-383-2204, Fax: +81-75-383-2201, E-mail: tanaka@kuee.kyoto-u.ac.jp

The precedence-constrained single-machine problem to minimize total weighted completion time ($1|\text{prec}|\sum w_j C_j$) is well studied in the literature [1–13] and instances with up to 100 jobs were optimally solved even more than twenty years ago [3], although it is known to be strongly NP-hard [2]. In contrast, the precedence-constrained single-machine scheduling problem to minimize more general additive costs has not been studied extensively so far. In [14, 15], labeling procedures necessary for implementing dynamic programming to solve $1|\text{prec}|\sum f_j(t)$ were proposed. In the numerical experiment of [15], instances of the precedence-constrained single-machine total unweighted tardiness problem ($1|\text{prec}|\sum T_i$) and total weighted tardiness ($1|\text{prec}|\sum w_i T_i$) problem were solved by the dynamic programming. However, these instances were of the ordinary problems without precedence constraints, and job dominance properties in [16] were used as the precedence constraints. In other words, the purpose of the numerical experiment was to demonstrate the effectiveness of the dominance properties for improving the dynamic programming to solve the ordinary total tardiness problem without precedence constraints. It is true that they are applicable to the problem with precedence constraints that do not originate in dominance properties, but it cannot be expected that they work well especially when precedence constraints are not so restrictive and the number of feasible solutions (job sequences) is still very large. As is a well-known fact, strong dominance properties hold for $1||\sum T_i$ and the problem is even decomposable into subproblems [17]. On the other hand, only weak dominance properties have been proved for $1||\sum w_i T_i$. The numerical experiment in [15] seems to reflect this fact: Instances with 50 jobs were solved for $1||\sum T_i$, but only those with 20 jobs were for $1||\sum w_i T_i$. The more recent study [18] follows this line of research. In [18], Tang et al. proposed a method to treat precedence constraints in a Lagrangian relaxation-based lower bound of the single-machine total weighted tardiness problem. Then, the effect of taking into account the precedence constraints derived from the dominance properties was examined by a numerical experiment.

This paper is on an exact algorithm for the precedence-constrained single-machine scheduling problem to minimize general additive costs. The precedence constraints considered in this paper are not those derived from the dominance properties that appear in [15, 18], but more general ones as in [3]. To the best of the authors' knowledge, there has been no study on exact algorithms that consider such general precedence constraints directly for objective functions other than total weighted completion time, although the DP in [15] is applicable as already explained. For the single-machine problem without precedence constraints to minimize general additive objective functions, the first author and his col-

leagues have already proposed efficient exact algorithms that can solve instances with up to 300 jobs when machine idle time is forbidden [19] and with up to 200 jobs when it is permitted [20]. These results indicate that our algorithms outperform the other exact algorithms proposed so far for these classes of problems without precedence constraints. Therefore, it is expected that our framework is also effective for the precedence-constrained problem. Hence, this study will extend these algorithms to the precedence-constrained problem, but only the case when machine idle time is forbidden is considered. Numerical experiments will show that the proposed algorithm can solve all 50 jobs instances of the precedence-constrained total weighted tardiness problem ($1|prec|\sum w_i T_i$) and of the precedence-constrained total weighted earliness-tardiness problem without idle times ($1|prec, noidle|\sum(\alpha_i E_i + \beta_i T_i)$), and most of 100 jobs instances of $1|prec|\sum w_i T_i$.

The remainder of this paper is organized as follows. First, in Section 2, the problem considered in this study is formulated as a constrained shortest path problem. Next, in Section 3, our previous algorithms in [19, 20] are reviewed and how to extend them to the problem with precedence constraints are shortly stated. Then, in Section 4, Lagrangian relaxations necessary for constructing the proposed algorithm are given and then how to solve them by dynamic programming is explained. In Section 5, the proposed algorithm is summarized. Section 6 shows some numerical results, and finally, Section 7 concludes this study.

2. Problem Description and Network Representation

In this section the precedence-constrained single-machine scheduling problem treated in this study is described formally and it is formulated as a constrained shortest path problem.

Suppose that n jobs (job 1, job 2, ..., job n) are to be processed on a single machine that can process at most one job at a time. An integer processing time $p_i > 0$ is given for each job $i \in \mathcal{N} = \{1, 2, \dots, n\}$. An integer cost function $f_i(t)$ ($t \geq p_i$) is also given for each job i and the cost $f_i(C_i)$ is incurred when job i is completed at C_i . On some pairs of jobs i and j ($i, j \in \mathcal{N}$, $i \neq j$), the precedence constraint that job i should precede job j ($i \rightarrow j$) is imposed. All the precedence constraints are specified by an acyclic directed graph $\bar{G}_P = (V_P, \bar{A}_P)$ with $V_P = \{1, 2, \dots, n\}$ such that $i \rightarrow j$ when $(i, j) \in \bar{A}_P$. \bar{A}_P is assumed to be transitively closed: if $(i, k), (k, j) \in \bar{A}_P$, then $(i, j) \in \bar{A}_P$. The minimal expression of the precedence constraints is given by $G_P = (V_P, A_P)$, where $A_P \subseteq \bar{A}_P$ is the transitive reduction of \bar{A}_P . Preemption and machine idle time are forbidden and hence all the jobs should

be processed in the interval $[0, T]$ where $T = \sum_{i \in \mathcal{N}} p_i$. The objective is to find a job sequence that satisfies the precedence constraints and that minimizes total job completion cost $\sum_{i \in \mathcal{N}} f_i(C_i)$.

We treat this problem as a constrained shortest path problem on an acyclic weighted directed graph $G = (V, A, W)$. Let us introduce a dummy job $n+1$ with $p_{n+1} = 1$, $f_{n+1}(t) \equiv 0$ that should be completed twice at time 0 and $T+1$ to introduce the source node $v_{n+1,0}$ and the sink node $v_{n+1,T+1}$ ¹. Next, define a node set V by

$$V = \{v_{n+1,0}\} \cup V_O \cup \{v_{n+1,T+1}\}, \quad (1)$$

$$V_O = \{v_{it} \mid i \in \mathcal{N}, p_i + \sum_{(j,i) \in \bar{A}_P} p_j \leq t \leq T - \sum_{(i,j) \in \bar{A}_P} p_j\}. \quad (2)$$

An arc set A is defined by

$$A = \{(v_{i,t-p_j}, v_{jt}) \mid v_{i,t-p_j}, v_{jt} \in V\}, \quad (3)$$

where the weight (length) $W(e)$ of an arc $e = (v_{i,t-p_j}, v_{jt}) \in A$ is given by $f_j(t)$. Then, our problem is equivalent to find a shortest path from the source node $v_{n+1,0}$ to the sink node $v_{n+1,T+1}$ such that v_{it} is visited exactly once for any $i \in \mathcal{N}$ and that v_{is} and v_{jt} in the path satisfy $s \leq t - p_j$ if $(i, j) \in A_P$. The optimal objective value is identical to the shortest path length, and v_{it} visited in the shortest path corresponds to the completion of job i at t in an optimal solution. Figure 1 shows an example of G for an instance with $n = 5$. The processing times are given by $p_1 = 1, p_2 = 2, p_3 = 2, p_4 = 3, p_5 = 1$ and a precedence constraint is on jobs 1 and 2 so that $1 \rightarrow 2$.

Hereafter, we use the following notation and definitions. A set of nodes visited in a path from $v_{n+1,0}$ to $v_{n+1,T+1}$ is denoted by \mathcal{P} , and the path corresponding to \mathcal{P} is referred to as “path \mathcal{P} ” for simplicity. Let $L(\mathcal{P})$ be the length of a path \mathcal{P} . Namely, $L(\mathcal{P})$ is defined by

$$L(\mathcal{P}) = \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} f_i(t). \quad (4)$$

For each job i , the number of occurrences of v_{it} in \mathcal{P} is denoted by $\mathcal{V}_i(\mathcal{P})$, which is defined by

$$\mathcal{V}_i(\mathcal{P}) = |\{v_{it} \mid v_{it} \in \mathcal{P}\}|. \quad (5)$$

¹In [19], another dummy job 0 is introduced and the source node is denoted by v_{00} . However, only one dummy node is introduced here according to the newer notation in [20]. In [20], job 0 denotes a unit idle time.

Then, the constraints that a path \mathcal{P} should visit v_{it} ($i \in \mathcal{N}$) exactly once can be written by

$$\mathcal{V}_i(\mathcal{P}) = 1, \quad i \in \mathcal{N}. \quad (6)$$

In addition, the precedence constraints can be expressed by

$$s \leq t - p_j, \quad (i, j) \in A_P, \quad v_{is}, v_{jt} \in \mathcal{P}, \quad (7)$$

or, equivalently,

$$\sum_{v_{is} \in \mathcal{P}} s \leq \sum_{v_{jt} \in \mathcal{P}} t - p_j, \quad (i, j) \in A_P. \quad (8)$$

We denote the set of all the feasible paths satisfying these constraints by \mathcal{Q} . Then, our problem (P) can be formulated as follows.

$$(P) : \min_{\mathcal{P}} L(\mathcal{P}) \quad \text{s.t. } \mathcal{P} \in \mathcal{Q}. \quad (9)$$

3. Previous Algorithm Overview and Outline of Its Extension

The proposed algorithm is based on the SSDP (Successive Sublimation Dynamic Programming) method [21, 22] as our previous algorithms [19, 20]. This algorithm is similar to the cutting plane algorithm: It first computes a lower bound by a simple relaxation of the original problem and then improves it successively by adding constraints (cuts) to the relaxation until the gap between the lower and upper bounds vanishes. Since all the relaxations are solved by dynamic programming, it is inevitable that the number of dynamic programming states increases as the number of added constraints increases. To cope with it, unnecessary states are eliminated in the course of the algorithm, which is the key point of the SSDP method. In our previous studies [19, 20], several improvements for the original algorithm in [22] were proposed not only to reduce unnecessary states, but also to improve the efficiency of the algorithm. These improvements made the algorithm outperform the existing algorithms and instances even with 300 jobs could be solved optimally. This section gives a brief sketch of our previous algorithms. Then how to extend it to our problem and where its difficulty lies will be explained shortly.

The following Lagrangian relaxations are employed in [19, 20]. First, the constraints on the number of job occurrences (6) are relaxed by penalizing their violations. Next, the constraints on successive jobs [24] are added to the relaxation. These constraints forbid such a sub-path in the network representation that

nodes representing the same job are visited twice or more. Therefore, they correspond to the cycle elimination constraints that often appear in routing problems (e.g. [23]) Then, the constraints based on the dominance of adjacent pairs of jobs [25, 19] are added. These constraints forbid such a processing order of two adjacent jobs that interchanging their order can improve the solution. Finally, the relaxed constraints (6) are gradually recovered until the gap between the lower and upper bounds vanishes. An upper bound is computed in the course of the algorithms by first converting a solution of a relaxation to a feasible solution of the original problem and then by applying the dynasearch [29, 30].

All these relaxations are solvable by dynamic programming algorithms. However, the number of dynamic programming states, which corresponds to the size of the underlying network, increases exponentially as the constraints (6) are recovered. Therefore, unnecessary dynamic programming states are eliminated in the course of the SSDP method. In the original algorithm [22], a lower bound to pass through each state is computed by applying dynamic programming in both forward and backward directions, and the state is eliminated when the lower bound is not smaller than the current upper bound. This corresponds to the elimination of a node or an arc in the network representation. In addition to this, the network reduction based on the dominance of more than two successive jobs, the constraint propagation technique, and the compressed network representation were introduced in [20] to reduce the network more.

In this paper, we will extend the algorithm in [19] improved by integrating the techniques in [20] to the problem with precedence constraints. Roughly speaking, the following are necessary for this purpose:

- (a) Relaxations for the problem with precedence constraints that are efficiently solvable by dynamic programming.
- (b) Extension of the network reduction techniques.
- (c) Extension of the upper bound computation method.

Among these, (b) is almost obvious and (c) is not difficult, while (a) is not direct. It is true that we can derive a relaxation of the original problem (P) by penalizing the violations of the precedence constraints (7) or (8) as well as the violations of the constraints (6). However, to construct an SSDP algorithm, they should be recovered again to the relaxation and the obtained relaxation should be solvable by dynamic programming. One of our main contributions is that it is shown to be solvable by simple modifications to the previous dynamic programming algorithm without increasing the time complexity.

In the next section, we will give the relaxations in the proposed algorithm and show how to solve them by dynamic programming. Then, in Section 5, the proposed algorithm will be summarized together with (c).

4. Lagrangian Relaxations in the Proposed Algorithms

This section gives the relaxations used in the proposed algorithm. The primary difference from those in the previous algorithms [19, 20] is, as already explained in the preceding section, that the violation of all or part of the precedence constraints is penalized by the Lagrangian relaxation technique. However, it does not follow that the original dynamic programming is applicable to the relaxations considered here. In the following, the dynamic programming to solve the relaxations will be stated.

4.1. Lagrangian relaxation of (6) and (8)

Let us penalize the constraints on the number of occurrences (6) by μ_i^O ($i \in \mathcal{N}$) and the precedence constraints of the form (8) by $\mu_{ij}^P \geq 0$ ($(i, j) \in A_P$). Then, we obtain the following relaxation (LR₀):

$$(\text{LR}_0) : \min_{\mathcal{P}} \left\{ L(\mathcal{P}) + \sum_{i \in \mathcal{N}} \mu_i^O (1 - \mathcal{V}_i(\mathcal{P})) + \sum_{(i,j) \in A_P} \mu_{ij}^P \left(\sum_{v_{is} \in \mathcal{P}} s - \sum_{v_{jt} \in \mathcal{P}} t + p_j \right) \right\}. \quad (10)$$

By introducing ω_i ($i \in \mathcal{N}$) defined by

$$\omega_i = \sum_{(j,i) \in A_P} \mu_{ji}^P - \sum_{(i,j) \in A_P} \mu_{ij}^P, \quad (11)$$

(10) can be rewritten as

$$(\text{LR}_0) : \min_{\mathcal{P}} L_R(\mathcal{P}; \boldsymbol{\mu}^O, \boldsymbol{\mu}^P) + \sum_{i \in \mathcal{N}} \mu_i^O + \sum_{(i,j) \in A_P} \mu_{ij}^P p_j, \quad (12)$$

where $L_R(\mathcal{P}; \boldsymbol{\mu}^O, \boldsymbol{\mu}^P)$ is defined by

$$\begin{aligned} L_R(\mathcal{P}; \boldsymbol{\mu}^O, \boldsymbol{\mu}^P) &= L(\mathcal{P}) - \sum_{i \in \mathcal{N}} \mu_i^O \mathcal{V}_i(\mathcal{P}) + \sum_{(i,j) \in A_P} \mu_{ij}^P \left(\sum_{v_{is} \in \mathcal{P}} s - \sum_{v_{jt} \in \mathcal{P}} t \right) \\ &= \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} f_i(t) - \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} \mu_i^O - \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} \omega_i t \\ &= \sum_{\substack{v_{it} \in \mathcal{P} \\ i \in \mathcal{N}}} (f_i(t) - \mu_i^O - \omega_i t). \end{aligned} \quad (13)$$

From (13), we can see that (LR_0) for a set of multipliers μ^O and μ^P is equivalent to the unconstrained shortest path problem on $G = (V, A, W')$, where $W'(e)$ is defined by

$$W'((v_{i,t-p_j}, v_{jt})) = f'_j(t) = f_j(t) - \mu_j^O - \omega_{jt}. \quad (14)$$

The relaxation (LR_0) is the same as that of the problem without precedence constraints except the weights of the arcs, and hence can be solved by dynamic programming in $O(nT)$ time (see [24]).

4.2. Improvement by the constraints on successive jobs

To improve the lower bound obtained by (LR_0) , the following constraints are imposed on (LR_0) as in the previous algorithms [22, 19, 20]:

For any $i \in \mathcal{N}$, nodes representing job i , i.e., v_{it} should not be visited more than once in any $\lambda + 1 > 0$ successive nodes in a path. (15)

These constraints forbid such a sub-path that $v_{i,t-p_i} \rightarrow v_{it}$ when $\lambda = 1$ and $v_{i,t-p_i-p_j} \rightarrow v_{j,t-p_j} \rightarrow v_{it}$ additionally when $\lambda = 2$. A set of paths satisfying these constraints on successive nodes is denoted by \mathcal{Q}_λ , and the relaxation with the constraints is denoted by (LR_λ) . Here, we only consider (LR_1) and (LR_2) as in [19, 20]. By defining

$$A_S = A \setminus \{(v_{i,t-p_i}, v_{it}) \mid v_{i,t-p_i}, v_{it} \in V_O\}, \quad (16)$$

we introduce a reduced network $G_S = (V, A_S, W')$. Then, (LR_1) becomes equivalent to the unconstrained shortest path problem on G_S . On the other hand, (LR_2) remains the constrained shortest path problem even on G_S . Since the precedence constraints are relaxed, the time complexities of (LR_1) and (LR_2) by dynamic programming do not differ from those in [19] and are given by $O(nT)$ and $O(n^2T)$, respectively. It is worth noting that the time complexity of (LR_2) is the same as the number of arcs $|A_S|$ in G_S because each arc is evaluated once in the dynamic programming. For the detailed recursive equations, please refer to [19].

Figure 2 shows G_S derived from G in Fig. 1. In G_S , the arcs $(v_{i,t-p_i}, v_{it})$ are eliminated for all i and t from G .

4.3. Improvement by the constraints on adjacent pairs of jobs

The second constraints are derived from the dominance theorem of dynamic programming [26] for adjacent pairs of jobs [25, 19, 20]. For the problem without precedence constraints, they restrict processing orders of adjacent pairs of jobs by checking their total costs. Here, we should slightly modify them to ensure the consistency with the precedence constraints.

Suppose that two jobs i and j ($i, j \in \mathcal{N}$, $i \neq j$) are processed adjacently. If $(i, j) \in \bar{A}_P$ and $(i, j) \notin A_P$ (or, $(j, i) \in \bar{A}_P$ and $(j, i) \notin A_P$), this situation cannot happen because there should exist another job k satisfying $(i, k), (k, j) \in \bar{A}_P$ (or, $(j, k), (k, i) \in \bar{A}_P$). In this case, we forbid the two jobs i and j to be processed adjacently. On the other hand, if $(i, j) \in A_P$ or $(j, i) \in A_P$, the two jobs can be processed adjacently, but their processing order is restricted by the precedence constraint. If no precedence constraint exists between the jobs i and j , i.e., $(i, j), (j, i) \notin \bar{A}_P$, we compare the total completion costs of the two jobs when they are completed at t : $f_i(t - p_j) + f_j(t)$ when job i precedes job j , and $f_j(t - p_i) + f_i(t)$ when job j precedes job i . Then, the processing order that yields the larger cost is forbidden. If the two costs are identical, either (but not arbitrary) processing order can be forbidden without loss of optimality. To summarize, the processing orders of any adjacent pairs of jobs can be restricted and these restrictions are imposed on (LR_2) as constraints. This yields a new relaxation (\widehat{LR}_2) .

In the network representation, these adjacency constraints eliminate from G_S , those edges corresponding to the forbidden processing orders. Thus, we define $\widehat{G}_S = (V, \widehat{A}_S, W')$ by

$$\widehat{A}_S = A_S \setminus \{(v_{i,t-p_j}, v_{jt}) \mid \text{job } j \text{ cannot be preceded adjacently by job } i \text{ at } t\}. \quad (17)$$

Then, (\widehat{LR}_2) becomes equivalent to the shortest path problem on \widehat{G}_S under the constraints (15) on three successive nodes (for $\lambda = 2$). The time complexity of (\widehat{LR}_2) does not increase from (LR_2) and it can be solved in $O(n^2T)$ time because $\widehat{G}_S \subset G_S$ (and $|\widehat{A}_S|$ is $O(n^2T)$). Figure 3 shows an example of \widehat{G}_S derived from G_S in Fig. 2 under some dominance properties on adjacent pairs of jobs. For example, we assume that the processing order $3 \rightarrow 1$ is better than $1 \rightarrow 3$ if they are completed before $t = 6$ and the converse is true when they are completed after $t = 7$. Hence $(v_{1,t-2}, v_{3t})$ are eliminated for $t \leq 6$ and $(v_{3,t-1}, v_{1t})$ are eliminated for $t \geq 7$. In addition, $(v_{2,t-1}, v_{1t})$ are eliminated for all t due to the precedence constraint $1 \rightarrow 2$.

4.4. Recovering the relaxed constraints

In our previous algorithms without precedence constraints [19, 20], the relaxed constraints (6) are gradually recovered to (\widehat{LR}_2) . More specifically, for some $\mathcal{M} = \{\phi_1, \dots, \phi_m\} \subseteq \mathcal{N}$,

$$\mathcal{V}_{\phi_k}(\mathcal{P}) = 1, \quad 1 \leq k \leq m, \quad (18)$$

are imposed on the relaxation corresponding to (\widehat{LR}_2) . It is easy to see that the relaxation approaches the original problem as $m = |\mathcal{M}|$ increases and it becomes

equivalent when $m = n$. Hence, the lower bound obtained by solving the relaxation improves as the relaxed constraints are recovered, and the gap between the lower and upper bounds vanishes in a finite number of iterations. To solve the relaxation in each iteration, it takes $O(n^2 2^m T)$ time [19].

In this study some of the precedence constraints (7), i.e.,

$$s \leq t - p_i, \quad (\phi_k, i) \in A_P, \quad v_{\phi_k s}, v_{it} \in \mathcal{P}, \quad 1 \leq k \leq m, \quad (19)$$

$$s \leq t - p_{\phi_k}, \quad (i, \phi_k) \in A_P, \quad v_{is}, v_{\phi_k t} \in \mathcal{P}, \quad 1 \leq k \leq m, \quad (20)$$

are recovered at the same time. In addition, μ_{ij}^P is set to zero if $i, j \in \mathcal{M}$. Clearly, the relaxation generated in this manner, which is referred to as $(\widehat{\text{LR}}_2^m)$, becomes equivalent to (P) when $\mathcal{M} = \mathcal{N}$. Therefore, an optimal solution of (P) is obtainable in a finite number of iterations by solving $(\widehat{\text{LR}}_2^m)$ with m increased until the gap between the lower and upper bounds vanishes (in practice, the algorithm can be terminated the gap becomes less than one because the optimal objective value of the original problem (P) is integral). Our claim here is that $(\widehat{\text{LR}}_2^m)$ is solvable in $O(n^2 2^m T)$ time and hence the time complexity does not increase even under the existence of the precedence constraints (19) and (20). In the following, it will be shown step by step.

First, assume that only the constraints (18) are imposed on $(\widehat{\text{LR}}_2)$ and denote the resulting relaxation by $(\widetilde{\text{LR}}_2^m)$. To solve it, define an m -dimensional vector \mathbf{q}_i^m ($i \in \mathcal{N}$) by $\mathbf{q}_i^m = (q_{i1}, \dots, q_{im})$, where

$$q_{ik} = \begin{cases} 1 & i = \phi_k, \\ 0 & \text{otherwise.} \end{cases} \quad (21)$$

Let us also define an m -dimensional vector \mathbf{q}_{n+1}^m by $\mathbf{q}_{n+1}^m = (0, \dots, 0)$. Next, we introduce a weighted directed graph $G_S^m = (V^m, A_S^m, W^l)$. The node set V^m is defined by

$$V^m = \{v_{n+1,0}^{\mathbf{0}_m}\} \cup V_O^m \cup \{v_{n+1,T+1}^{\mathbf{1}_m}\}, \quad (22)$$

$$V_O^m = \{v_{it}^{\mathbf{b}} \mid v_{it} \in V_O, \mathbf{q}_i^m \leq \mathbf{b} \leq \mathbf{1}_m\}, \quad (23)$$

where $\mathbf{0}_m$ and $\mathbf{1}_m$ denote m -dimensional vectors whose elements are all zero and all one, respectively. The arc set A_S^m is defined by

$$A_S^m = \{(v_{i,t-p_j}^{\mathbf{b}-\mathbf{q}_j^m}, v_{jt}^{\mathbf{b}}) \mid (v_{i,t-p_j}, v_{jt}) \in \widehat{A}_S, \mathbf{q}_i^m + \mathbf{q}_j^m \leq \mathbf{b} \leq \mathbf{1}_m\}. \quad (24)$$

Then, the shortest path problem on G_S^m under the constraints (15) for $\lambda = 2$ is equivalent to (\widetilde{LR}_2^m) . To solve it by (forward) dynamic programming, the constrained shortest path from the source node $v_{n+1,0}^{0_m}$ to v_{jt}^b through $(v_{i,t-p_j}^{b-q_j^m}, v_{jt}^b)$ is recursively computed. Since, in this shortest path, the number of occurrences of the nodes representing job ϕ_k ($1 \leq k \leq m$) is equal to the k -th element of \mathbf{b} , it is ensured that $v_{\phi_k t}^b$ appears exactly once for all k ($1 \leq k \leq m$) in the obtained shortest path from $v_{n+1,0}^{0_m}$ to $v_{n+1,T+1}^{1_m}$. The time complexity is given by $O(n^2 2^m T)$ because $|A^m|$ is $O(n^2 2^m T)$ ($|V^m|$ is $O(n 2^m T)$) and the in-degree of each node is at most n .

Figure 4 gives G_S^m derived from \widehat{G}_S in Fig. 3 for $\mathcal{M} = \{1\}$ and $m = 1$. It can be verified from the structure of the network that any path cannot reach the sink node $v_{6,10}^1$ without passing through v_{1t}^1 exactly once for some t . However, the path shown in thick arrows, $v_{6,0}^0 \rightarrow v_{5,1}^0 \rightarrow v_{2,3}^0 \rightarrow v_{3,5}^0 \rightarrow v_{1,6}^1 \rightarrow v_{4,9}^1 \rightarrow v_{6,10}^1$, does not satisfy the precedence constraint because it passes through $v_{2,3}^0$ before $v_{1,6}^1$.

Next, consider (\widetilde{LR}_2^m) , i.e., (\widehat{LR}_2^m) with the precedence constraints (19) and (20). As we have already seen, the number of occurrences of the nodes representing job ϕ_k in the path from $v_{n+1,0}^{0_m}$ to v_{jt}^b is equal to b_k . It implies that this path breaks the precedence constraint (19) if $(\phi_k, j) \in A_P$ and $b_k = 0$. Similarly, it breaks the constraint (20) if $(j, \phi_k) \in A_P$ and $b_k = 1$. Therefore, we can satisfy the precedence constraints (19) and (20), only by removing from G_S^m all the nodes v_{jt}^b with such \mathbf{b} . Clearly, the time complexity of the constrained shortest path problem on G_S^m does not change even when some nodes are removed. Hence, we can conclude that (\widehat{LR}_2^m) is solvable in $O(n^2 2^m T)$ time.

Notice that G_S^m with $\phi_k \in \mathcal{M}$ only forbids $\phi_k \rightarrow i$ under the precedence constraint $i \rightarrow \phi_k$ (or $i \rightarrow \phi_k$ under $\phi_k \rightarrow i$). Since it is possible that a node representing job i does not appear in a feasible path from $v_{n+1,0}^{0_m}$ to $v_{n+1,T+1}^{1_m}$ if $i \notin \mathcal{M}$, we cannot say that $i \rightarrow \phi_k$ (or $\phi_k \rightarrow i$) is satisfied in this case. However, $i \rightarrow j$ for any $(i, j) \in A_P$ is satisfied when $\mathcal{M} = \mathcal{N}$ because the constraints (6) are all recovered and hence all the jobs should appear exactly once in any feasible path.

We can apply the above arguments not only to the minimal precedence constraints ($i \rightarrow j$ for $(i, j) \in A_P$), but also to the transitive closure ($i \rightarrow j$ for $(i, j) \in \overline{A_P}$), which improves the lower bound and reduces the network more. To summarize, we are to solve the shortest path problem on a directed graph $\widehat{G}_S^m =$

$(\widehat{V}^m, \widehat{A}_S^m, W')$ under the constraints (15) for $\lambda = 2$, where

$$\widehat{V}^m = V^m \setminus \left(\{v_{it}^b \mid \exists k, (\phi_k, i) \in \bar{A}_P, b_k = 0\} \cup \{v_{it}^b \mid \exists k, (i, \phi_k) \in \bar{A}_P, b_k = 1\} \right), \quad (25)$$

$$\widehat{A}_S^m = \{(v_{i,t-p_j}^{b-q_j^m}, v_{jt}^b) \mid (v_{i,t-p_j}^{b-q_j^m}, v_{jt}^b) \in A_S^m, v_{i,t-p_j}^{b-q_j^m}, v_{jt}^b \in \widehat{V}^m\}. \quad (26)$$

Figure 5 shows \widehat{G}_S^m derived from G_S^m in Fig. 4, where v_{2t}^0 are eliminated for all t . It is obvious that this network ensures that $v_{2\bullet}^0$ is never visited before job $v_{1\bullet}^1$. The path shown in thick arrows, $v_{6,0}^0 \rightarrow v_{3,2}^0 \rightarrow v_{1,3}^1 \rightarrow v_{5,4}^1 \rightarrow v_{4,7}^1 \rightarrow v_{3,9}^1 \rightarrow v_{6,10}^1$, is still feasible in \widehat{G}_S^m , where v_{2t}^1 is not visited for any t . However, it is eliminated or at least becomes infeasible when job 2 is added to \mathcal{M} .

4.5. Improvement by slack variable optimization

Since the precedence constraints (7) are inequality constraints, the lower bound can be improved by converting them to equality constraints with slack variables and then solving optimization problems with regard to the slack variables [4]. We apply this method to all the relaxations (LR_1) , (LR_2) and (\widehat{LR}_2^m) .

5. Proposed Algorithm

Our proposed algorithm utilizes the Lagrangian relaxations (LR_1) , (\widehat{LR}_2) and (\widehat{LR}_2^m) in the preceding section. The algorithm is composed of three stages: (LR_1) is solved first, (\widehat{LR}_2) is solved next, and then (\widehat{LR}_2^m) is solved with \mathcal{M} ($m = |\mathcal{M}|$) increased. These correspond to the main loop of the SSDP method that generates and solves better relaxations by adding constraints. Since the size of the network G_S^m corresponding to (\widehat{LR}_2^m) increases exponentially as the algorithm proceeds and constraints are recovered, it is crucial to suppress the increase of the network size as much as possible.

The algorithm is summarized as follows.

- Stage 1** An initial upper bound UB is computed. Then, the conjugate subgradient algorithm [27, 28] is applied to the Lagrangian dual corresponding to (LR_1) . The algorithm is terminated without entering the next stage if the gap between the best lower bound and UB is less than one.
- Stage 2** The multipliers are re-adjusted by the conjugate subgradient algorithm for the Lagrangian dual corresponding to (LR_2) . An upper bound is computed every 5 iterations and UB is updated if necessary. The algorithm

is terminated without entering the next stage if the gap between the best lower bound and UB is less than one.

Stage 3 The relaxation $(\widehat{\text{LR}}_2^m)$ is solved with \mathcal{M} increased from $\mathcal{M} = \emptyset$. When \mathcal{M} is increased, an upper bound is computed and UB is updated if necessary. Then, multipliers are re-adjusted by the conjugate subgradient algorithm for a smaller number of iterations than those in Stage 1 and Stage 2. After it terminates, new jobs are chosen from $\mathcal{N} \setminus \mathcal{M}$ and added to \mathcal{M} . It is repeated until the gap between the lower bound and UB becomes less than one.

Elimination of unnecessary dynamic programming states in the SSDP method is performed in the course of the algorithm by the network reduction techniques in [20], which were explained in Section 3. These are applicable to our problem with minor modifications. An upper bound is computed also in a similar way to that in [20]: A solution of a relaxation is converted to a feasible solution of (P), and then the enhanced dynasearch [29, 30] is applied to improve it. To convert to a feasible solution, we first remove all the jobs that break the precedence constraints or that occur more than once. Then, the jobs that do not appear in the partial schedule are inserted into it optimally or greedily so as not to break the precedence constraints. To insert the absent jobs optimally, the dynamic programming algorithm in [22] is applied with a modification to satisfy the precedence constraints. However, the time complexity of the algorithm depends exponentially on the number of absent jobs n' and is given by $O((n - n' + 1)n'2^{n'})$. Therefore, a heuristic algorithm is employed instead to insert the absent jobs one by one according to the SPT (shortest processing time) order, when the number of such jobs is large. It takes only $O(n' \log n')$ for ordering the absent jobs and $O((2n - n' + 1)n')$ for inserting them one by one (the number of candidate positions for the i -th absent job is $n - n' + i$). The obtained solution is improved by the enhanced dynasearch extended to our problem by forbidding moves in the neighborhood that break the precedence constraints. The dynasearch is a local search algorithm that employs a neighborhood called dynasearch neighborhood. It is defined by solutions obtained via any number of pairwise interchanges (PIs) that do not intersect with each other. In addition to PIs, extraction and forward shifted re-insertions (EFSRs), and extraction and backward shifted re-insertions (EFSRs) are considered in the enhanced dynasearch neighborhood. The primary advantage of these neighborhoods is that the best solution in the neighborhoods is obtainable in polynomial time by dynamic programming although they are composed of an exponential number of

solutions. To extend them to the problem with precedence constraints, we simply forbid such PIs, EFSRs and EFSRs that break the precedence constraints, and this does not increase the time complexity of the dynamic programming. The initial upper bound is computed in the same manner, except that the length of the partial schedule is zero and all the jobs are inserted greedily.

In [19, 20], the Lagrangian multipliers obtained in Stage 2 are also used in Stage 3, while in the proposed algorithm they are re-adjusted for (\widehat{LR}_2^m) in Stage 3. It is because the duality gap of our problem is larger than that of the problem without precedence constraints and hence it is necessary to improve the lower bound as much as possible. Another reason is that if jobs i and j appear in \mathcal{M} , μ_{ij}^P is set to zero to remove the corresponding penalty term from the objective function because the corresponding precedence constraint $i \rightarrow j$ is always satisfied. This affects the optimal Lagrangian multipliers much and thus justifies the re-adjustment.

The jobs to be added to \mathcal{M} in Stage 3 are determined as follows. For each job $i \in \mathcal{N} \setminus \mathcal{M}$,

$$\sum_{(j,i) \in A_P} \mu_{ji}^P + \sum_{(i,j) \in A_P} \mu_{ij}^P \quad (27)$$

is computed and a job with a larger (27) is chosen first. By doing this, precedence constraints that affect the objective value are supposed to be satisfied as early as possible. Ties are broken by the occurrences of corresponding nodes in \widehat{G}_S^m , and a job that occurs less frequently is chosen first.

6. Numerical Experiment

The proposed algorithm is applied to the precedence-constrained single-machine total weighted tardiness problem ($1|\text{prec}|\sum w_i T_i$) and the precedence-constrained single-machine total weighted earliness-tardiness problem without machine idle time ($1|\text{prec}, \text{noidle}|\sum(\alpha_i E_i + \beta_i T_i)$). Instances are generated from the OR-Library instances of $1||\sum w_i T_i$ with 40, 50 and 100 jobs that are available from <http://people.brunel.ac.uk/~mastjjb/jeb/orlib/wtinfo.html>. In the OR-Library instances, the processing time p_i , the tardiness weight w_i and the due date d_i of each job i were generated from the integer uniform distributions $[1, 100]$ and $[1, 10]$, $[T(1 - \text{TF} - \text{RDD}/2), T(1 - \text{TF} + \text{RDD}/2)]$, respectively. Here, TF and RDD are the tardiness factor and the range of due dates, respectively, and they are chosen from $\{0.2, 0.4, 0.6, 0.8, 1.0\}$. There are five instances for each combination of n , TF and RDD, and hence 125 instances for each n . To generate instances of $1|\text{prec}|\sum w_i T_i$ from these, precedence constraints are added as in [3, 4]: For every pair of i and j

($i, j \in \mathcal{N}, i < j$), whether the precedence constraint $i \rightarrow j$ is imposed on or not is determined by a specified probability P . The instances of $1|\text{prec}, \text{noidle}|\sum(\alpha_i E_i + \beta_i T_i)$ are generated by adding precedence constraints in a similar way to the instances of $1|\text{noidle}|\sum(\alpha_i E_i + \beta_i T_i)$ used in [19], which were generated from the OR-Library instances by choosing the tardiness weight $\beta_i = w_i$ and by generating the earliness weight α_i from the integer uniform distribution $[1, 10]$. We choose P from $\{0.005, 0.01, 0.02, 0.05, 0.1, 0.2\}$ and thus there are 125 instances for each combination of n , P and the problem type. The computation is performed on a desktop computer with Core i7 960 (3.2GHz) CPU and 12GB RAM. The maximum memory size for storing the network structure is set to 3GB and the time limit is set to 10000s. For comparison, the dynamic programming in [15] is implemented and applied to these instances. Since this algorithm was originally proposed for employing dominance properties, the precedence relations derived from the dominance properties summarized in [31] are used for $1|\text{prec}|\sum w_i T_i$ additionally, as far as they are consistent with the original precedence constraints. In other words, the precedence relations are put into G_P as far as no cycle is formed. Note that these additional precedence constraints are available only for $1|\text{prec}|\sum w_i T_i$ because no such dominance properties are known for $1|\text{prec}, \text{noidle}|\sum(\alpha_i E_i + \beta_i T_i)$.

The results are shown in Tables 1 and 2, where average (ave) and maximum (max) CPU times over optimally solved instances (solved) are given in seconds for the proposed algorithm (Proposed) and the dynamic programming in [15] (DP in [15]). In “proposed”, the instances without precedence constraints ($P = 0.0$) are solved not by the proposed algorithm but by an improved version of the algorithm in [19]. From the tables, we can verify that the proposed algorithm successfully solved all the instances with 40 and 50 jobs, and almost all the instances with 100 jobs of $1|\text{prec}|\sum w_i T_i$. For the unsolved instances, the proposed algorithm reached the memory limit or the time limit. On the other hand, the dynamic programming in [15] failed to solve instances only due to the memory limit. The latter algorithm could not solve some instances with 40 jobs. In particular, it could not solve even a single instance of $1|\text{prec}, \text{noidle}|\sum(\alpha_i E_i + \beta_i T_i)$ when $n = 40$ and $P = 0.0, 0.005, 0.01, 0.02$, or 0.05 because dominance properties cannot be exploited for this type of problem. However, the precedence constraints are so restrictive when $P = 0.2$ and the algorithm worked better than the proposed algorithm. Nevertheless, it could not solve some of the instances with $n = 100$ and $P = 0.2$. For both the algorithms, the hardest instances seems those with $P = 0.02$ or $P = 0.05$.

Next, the lower bounds obtained in Stage 2 of the proposed algorithm and those by the method in [18] are compared for the instances with $n = 40$. The

Table 1: Computational Results for Benchmark Instances of $1|\text{prec}|\sum w_i T_i$

| n | P | Proposed | | | DP in [15] | | |
|-----|-------|------------|---------------|----------------|------------|--------------|-------------|
| | | solved | CPU time (s) | | solved | CPU time (s) | |
| | | | ave | max | | ave | max |
| 40 | 0.0 | 125 | 0.04 | 0.09 | 121 | 0.27 | 2.82 |
| | 0.005 | 125 | 0.46 | 2.33 | 111 | 0.52 | 6.40 |
| | 0.01 | 125 | 0.62 | 7.78 | 99 | 0.61 | 9.37 |
| | 0.02 | 125 | 2.13 | 58.84 | 61 | 0.41 | 4.12 |
| | 0.05 | 125 | 7.66 | 32.15 | 40 | 0.09 | 0.43 |
| | 0.1 | 125 | 12.78 | 35.83 | 93 | 0.06 | 0.47 |
| | 0.2 | 125 | 1.97 | 13.99 | 125 | 0.00 | 0.01 |
| 50 | 0.0 | 125 | 0.09 | 0.19 | 89 | 0.93 | 21.05 |
| | 0.005 | 125 | 0.96 | 11.65 | 61 | 0.63 | 5.81 |
| | 0.01 | 125 | 2.49 | 27.49 | 44 | 1.03 | 16.54 |
| | 0.02 | 125 | 8.54 | 133.29 | 24 | 0.06 | 0.63 |
| | 0.05 | 125 | 42.63 | 541.49 | 22 | 0.06 | 0.54 |
| | 0.1 | 125 | 41.65 | 164.99 | 40 | 0.04 | 0.17 |
| | 0.2 | 125 | 4.56 | 29.77 | 125 | 0.00 | 0.02 |
| 100 | 0.0 | 125 | 0.91 | 2.43 | 31 | 0.19 | 5.58 |
| | 0.005 | 125 | 53.92 | 3781.20 | 25 | 0.12 | 1.97 |
| | 0.01 | 124 | 72.06 | 3129.99 | 22 | 0.01 | 0.27 |
| | 0.02 | 123 | 341.60 | 9860.74 | 20 | 0.17 | 3.14 |
| | 0.05 | 121 | 907.46 | 8812.20 | 7 | 0.08 | 0.20 |
| | 0.1 | 125 | 151.37 | 1236.93 | 22 | 0.16 | 0.72 |
| | 0.2 | 125 | 6.41 | 37.06 | 115 | 0.01 | 0.04 |

method in [18] is to exploit precedence constraints in the Lagrangian relaxation based-lower bound obtained by relaxing the machine availability constraint. To apply it, precedence relations derived from the dominance properties in [31] are also used as precedence constraints and put into the precedence graph G_P . Since the method in [18] cannot treat precedence constraints that form an undirected cycle in G_P , such cycles are broken randomly.

The results are summarized in Tables 3 and 4, where the average (ave) and maximum (max) relative gaps are given in percent. Here, the relative gap is computed by $100((\text{optimal value}) - (\text{lower bound})) / (\text{optimal value})$ and when the absolute gap $(\text{optimal value}) - (\text{lower bound})$ is less than one, it is regarded as zero. From Table 3, we can see that much better lower bounds are obtained by our algo-

Table 2: Computational Results for Benchmark Instances of $1|prec, noidle|\sum(\alpha_i E_i + \beta_i T_i)$

| n | P | Proposed | | | DP in [15] | | |
|-----|-------|------------|---------------|----------------|------------|--------------|-------------|
| | | solved | CPU time (s) | | solved | CPU time (s) | |
| | | | ave | max | | ave | max |
| 40 | 0.0 | 125 | 0.06 | 0.11 | 0 | — | — |
| | 0.005 | 125 | 0.52 | 2.32 | 0 | — | — |
| | 0.01 | 125 | 1.21 | 10.47 | 0 | — | — |
| | 0.02 | 125 | 3.65 | 42.62 | 0 | — | — |
| | 0.05 | 125 | 17.36 | 68.98 | 0 | — | — |
| | 0.1 | 125 | 21.77 | 111.49 | 25 | 0.14 | 0.36 |
| | 0.2 | 125 | 4.17 | 15.33 | 125 | 0.01 | 0.03 |
| 50 | 0.0 | 125 | 0.11 | 0.22 | 0 | — | — |
| | 0.005 | 125 | 1.88 | 37.68 | 0 | — | — |
| | 0.01 | 125 | 5.67 | 45.52 | 0 | — | — |
| | 0.02 | 125 | 18.83 | 110.47 | 0 | — | — |
| | 0.05 | 125 | 108.92 | 1755.38 | 0 | — | — |
| | 0.1 | 125 | 81.50 | 394.23 | 1 | 0.06 | 0.06 |
| | 0.2 | 125 | 8.96 | 29.93 | 125 | 0.01 | 0.06 |
| 100 | 0.0 | 125 | 1.48 | 3.38 | 0 | — | — |
| | 0.005 | 123 | 224.12 | 1987.94 | 0 | — | — |
| | 0.01 | 110 | 1050.39 | 9581.78 | 0 | — | — |
| | 0.02 | 61 | 1140.46 | 9564.18 | 0 | — | — |
| | 0.05 | 40 | 1199.38 | 7709.55 | 0 | — | — |
| | 0.1 | 79 | 1649.30 | 9442.81 | 0 | — | — |
| | 0.2 | 125 | 32.56 | 230.23 | 5 | 0.02 | 0.04 |

rithm than the method in [18] for $1|prec|\sum w_i T_i$. On the other hand, the difference is smaller for $1|prec, noidle|\sum(\alpha_i E_i + \beta_i T_i)$ in Table 4, and the method in [18] yields rather better lower bounds on average when $P = 0.02$, although it cannot exploit any dominance properties for this class of problem. The reason why the method in [18] yields better lower bounds for $1|prec, noidle|\sum(\alpha_i E_i + \beta_i T_i)$ than for $1|prec|\sum w_i T_i$ will be that taking into account precedence constraints affects the objective value much because the completion cost of a job is zero only when it is just-in-time. In contrast, the completion cost of a job is zero for $1|prec|\sum w_i T_i$ when it is before the due date and hence precedence constraints do not affect the objective value compared to $1|prec, noidle|\sum(\alpha_i E_i + \beta_i T_i)$.

It is direct to show that the best lower bound obtained by (LR_0) and that by the

Lagrangian relaxation of the machine availability constraint in [18] are identical when precedence constraints do not exist. Therefore, the advantages of our lower bound are the constraints in 4.2 and 4.3, and the improvement by the method in [4], while the advantage of the method in [18] is that precedence constraints are not relaxed. Therefore, it was expected that the method in [18] yields better lower bounds as P increases, but the results were not so. One reason might be that the method in [18] cannot be applied to the problem with precedence constraints including cycles and hence such precedence constraints are broken randomly. It follows that precedence constraints that are actually taken into account do not increase much even if P becomes large. On the other hand, all the precedence constraints are penalized in our method and, moreover, exactly considered for every pair of jobs in an adjacent position. This advantage seems to dominate over the advantage of the method in [18]. Nevertheless, the difference between the lower bounds in Stage 2 of our algorithm and those in [18] is not large and it might be difficult to solve the instances of $1|\text{prec}, \text{noidle}|\sum(\alpha_i E_i + \beta_i T_i)$ with $n = 100$ and $P = 0.02$ even if we succeeded in integrating the method in [18] into our SSDP framework.

Tables 3 and 4 also suggest difficulties in obtaining good lower bounds for this type of problem. The gaps are large and widely distributed when $P \neq 0$ compared to those when $P = 0$. If a precedence constraint $i \rightarrow j$ is imposed on two jobs i and j that are processed in this order in an optimal solution for the problem without precedence constraints, it has no effect on the gap (when the corresponding Lagrangian multiplier is appropriately adjusted and is set to zero). Since the precedence constraints in the benchmark instances are generated randomly, the number of such precedence constraints also varies randomly. Nevertheless, it will not affect the gap much if precedence constraints are handled successfully when computing lower bounds. However, in practice, the gap is widely distributed and totally affected by the randomness of the precedence constraints, which together with the large gap implies the difficulties in obtaining good lower bounds.

7. Conclusion

This study constructed an efficient exact algorithm for the precedence-constrained single-machine scheduling problem by extending our previous algorithms for the problem without precedence constraints. Numerical experiments showed that it can solve all the instances with 50 jobs of $1|\text{prec}|\sum w_i T_i$ and $1|\text{prec}, \text{noidle}|\sum(\alpha_i E_i + \beta_i T_i)$, and most instances with 100 jobs of $1|\text{prec}|\sum w_i T_i$, although CPU time is not so impressive for larger instances. To improve the algorithm, it seems necessary

Table 3: Tightness of Lower Bounds for $1|\text{prec}|\sum w_i T_i$ ($n = 40$)

| P | Stage 2 (%) | | Tang et al. [18] (%) | |
|-------|-------------|--------|----------------------|--------|
| | ave | max | ave | max |
| 0.0 | 0.00 | 0.00 | 0.75 | 13.89 |
| 0.005 | 0.04 | 2.83 | 1.59 | 28.45 |
| 0.01 | 0.43 | 43.19 | 3.64 | 35.17 |
| 0.02 | 1.11 | 27.64 | 7.47 | 90.93 |
| 0.05 | 6.03 | 100.00 | 20.21 | 100.00 |
| 0.1 | 6.94 | 57.04 | 34.81 | 100.00 |
| 0.2 | 1.24 | 13.87 | 33.85 | 81.37 |

Table 4: Tightness of Lower Bounds for $1|\text{prec, noidle}|\sum(\alpha_i E_i + \beta_i T_i)$ ($n = 40$)

| P | Stage 2 (%) | | Tang et al. [18] (%) | |
|-------|-------------|-------|----------------------|-------|
| | ave | max | ave | max |
| 0.0 | 0.04 | 2.71 | 1.35 | 9.05 |
| 0.005 | 0.25 | 8.75 | 1.27 | 8.19 |
| 0.01 | 0.76 | 7.92 | 1.09 | 5.94 |
| 0.02 | 1.69 | 14.41 | 1.34 | 22.80 |
| 0.05 | 4.51 | 18.14 | 5.46 | 31.74 |
| 0.1 | 5.64 | 22.67 | 17.71 | 48.74 |
| 0.2 | 1.72 | 7.42 | 26.25 | 51.57 |

to reduce the size of the network more by improving the lower bound via imposing some new constraints on the relaxations. It will also be necessary to extend the algorithm for the problem with machine idle time permitted. These are left for future research.

Another direction of research will be to construct a branch-and-bound algorithm. Although the SSDP framework is much more efficient than branch-and-bound algorithms [26, 32] for the problem without precedence constraints, a branch-and-bound scheme may work better when the number of imposed precedence constraints is large because the search space is restricted. Therefore, it will be worthwhile to compare a branch-and-bound algorithm with the proposed algorithm.

Acknowledgement

This work was partially supported by Grant-in-Aid for Young Scientists (B) 19760273, from the Ministry of Education, Culture, Sports, Science and Technol-

ogy Japan.

- [1] Sidney, J.B. (1975). Decomposition algorithms for single-machine sequencing with precedence relations and deferral costs. *Operations Research*, 23, 283–298.
- [2] Lawler, E.L. (1978). Sequencing jobs to minimize total weighted completion time subject to precedence constraints. *Annals of Discrete Mathematics*, 2, 75–90.
- [3] Potts, C.N. (1985). A Lagrangean based branch and bound algorithm for single machine sequencing with precedence constraints to minimize total weighted completion time. *Management Science*, 31, 1300–1311.
- [4] Hoogeveen, J.A., & van de Velde, S.L. (1995). Stronger Lagrangian bounds by use of slack variables: applications to machine scheduling problems. *Mathematical Programming*, 70, 173–190.
- [5] van de Velde, S.L. (1995). Dual decomposition of a single-machine scheduling problem. *Mathematical Programming*, 69, 413–428.
- [6] Chekuri, C., & Motwani, R. (1999). Precedence constrained scheduling to minimize sum of weighted completion times on a single machine. *Discrete Applied Mathematics*, 98, 29–38.
- [7] Woeginger, G.J. (2003). On the approximability of average completion time scheduling under precedence constraints. *Discrete Applied Mathematics*, 131, 237–252.
- [8] Margot, F., Queyranne, M., & Wang, Y. (2003). Decompositions, network flows, and a precedence constrained single-machine scheduling problem. *Operations Research*, 51, 981–992.
- [9] Correa, J.R., & Schulz, A.S. (2005). Single-machine scheduling with precedence constraints. *Mathematics of Operations Research*, 30, 1005–1021.
- [10] Ambühl, C., Mastrolilli, M., & Svensson, O. (2006). Approximating precedence-constrained single machine scheduling by coloring. *Lecture Notes in Computer Science*, 4110, 15–26.

- [11] Ambühl, C., Mastrolilli, M., Mutsanas, N., & Svensson, O. (2007). Scheduling with precedence constraints of low fractional dimension. *Lecture Notes in Computer Science*, 4513, 130–144.
- [12] Ambühl, C., & Mastrolilli, M. (2009). Single machine precedence constrained scheduling is a vertex cover problem. *Algorithmica*, 53, 488–503.
- [13] Schulz, A.S., & Uhan, N.A. (2011). Near-optimal solutions and large integrity gaps for almost all instances of single-machine precedence-constrained scheduling. *Mathematics of Operations Research*, 36, 14–23.
- [14] Baker, K.R., & Schrage, L.E. (1978). Finding an optimal sequence by dynamic programming: An extension to precedence-related tasks. *Operations Research*, 26, 111–120.
- [15] Schrage, L., & Baker, K.R. (1978). Dynamic programming solution of sequencing problems with precedence constraints. *Operations Research*, 26, 444–449.
- [16] Emmons, H. (1969). One-machine sequencing to minimize certain function of job tardiness. *Operations Research*, 17, 701–715.
- [17] Lawler, E.L. (1977). A “pseudopolynomial” algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics*, 1, 331–342.
- [18] Tang, L., Xuan, H., & Liu, J. (2007). Hybrid backward and forward dynamic programming based Lagrangian relaxation for single machine scheduling. *Computers & Operations Research*, 34, 2625–2636.
- [19] Tanaka, S., Fujikuma, S., & Araki, M. (2009). An exact algorithm for single-machine scheduling without machine idle time. *Journal of Scheduling*, 12, 575–593.
- [20] Tanaka, S., & Fujikuma, S. (2012). A dynamic-programming-based exact algorithm for single-machine scheduling with machine idle time. *Journal of Scheduling*, 15, 347–361.
- [21] Ibaraki, T. (1987). Enumerative approaches to combinatorial optimization. *Annals of Operations Research*, 10 and 11.

- [22] Ibaraki, T., & Nakamura, Y. (1994). A dynamic programming method for single machine scheduling. *European Journal of Operational Research*, 76, 72–82.
- [23] Christofides, N., Mingozzi, A., & Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations, *Mathematical Programming*, 20, 255–282.
- [24] Abdul-Razaq, T.S., & Potts, C.N. (1988). Dynamic programming state-space relaxation for single-machine scheduling. *Journal of the Operational Research Society*, 39, 141–152.
- [25] Sourd, F. (2009). New exact algorithms for one-machine earliness-tardiness scheduling. *INFORMS Journal on Computing*, 21, 167–175.
- [26] Potts, C.N., & Van Wassenhove, L.N. (1985). A branch and bound algorithm for the total weighted tardiness problem. *Operations Research*, 33, 363–377.
- [27] Sherali, H.D., & Ulular, O. (1989). A primal-dual conjugate subgradient algorithm for specially structured linear and convex programming problems. *Applied Mathematics and Optimization*, 20, 193–221.
- [28] Sherali, H.D., & Lim, C. (2007). Enhancing Lagrangian dual optimization for linear programs by obviating nondifferentiability. *INFORMS Journal on Computing*, 19, 3–13.
- [29] Congram, R.K., Potts, C.N., & van de Velde, S.L. (2002). An iterated dynasearch algorithm for the single-machine total weighted tardiness scheduling problem. *INFORMS Journal on Computing*, 14, 52–67.
- [30] Grosso, A., Della Croce, F., & Tadei, R. (2004). An enhanced dynasearch neighborhood for the single-machine total weighted tardiness scheduling problem. *Operations Research Letters*, 32, 68–72.
- [31] Abdul-Razaq, T.S. Potts, C.N., & Van Wassenhove, L.N. (1990). A survey of algorithms for the single machine total weighted tardiness scheduling problem. *Discrete Applied Mathematics*, 26, 235–253.
- [32] Pan, Y., & Shi, L. (2007). On the equivalence of the max-min transportation lower bound and the time-indexed lower bound for single-machine scheduling problems, *Mathematical Programming, Ser. A*, 543–559.

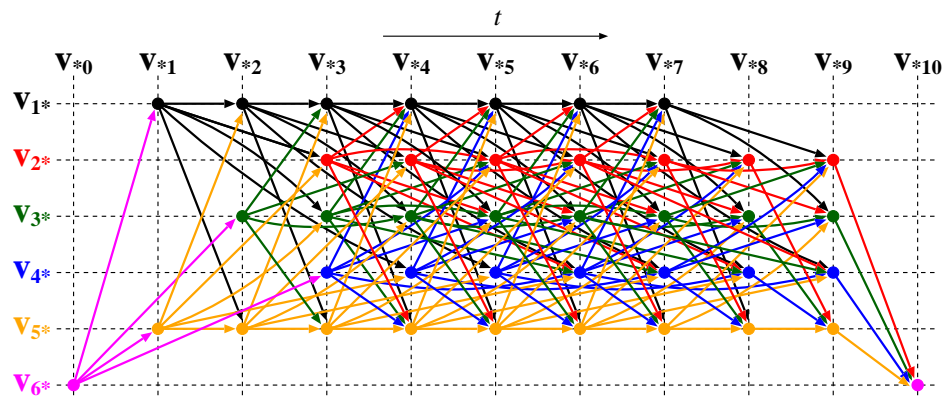


Figure 1: An Example of the Network G ($p_1 = 1$, $p_2 = 2$, $p_3 = 2$, $p_4 = 3$, $p_5 = 1$, a precedence constraint: $1 \rightarrow 2$)

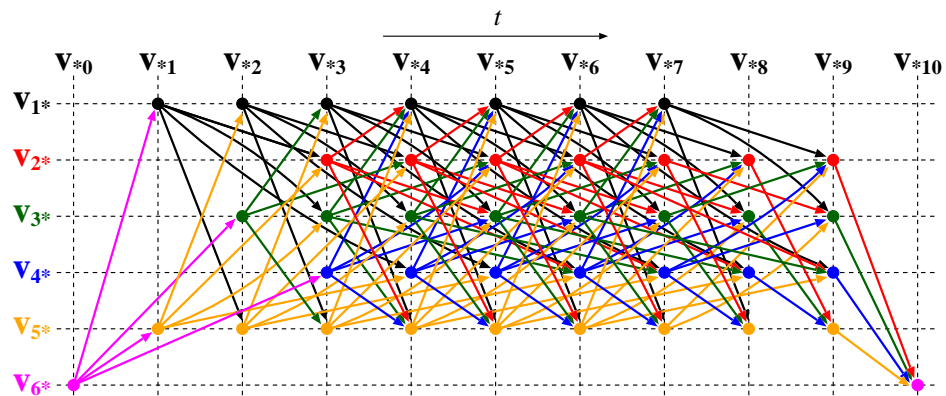


Figure 2: An Example of the Network G_S ($p_1 = 1$, $p_2 = 2$, $p_3 = 2$, $p_4 = 3$, $p_5 = 1$, a precedence constraint: $1 \rightarrow 2$)

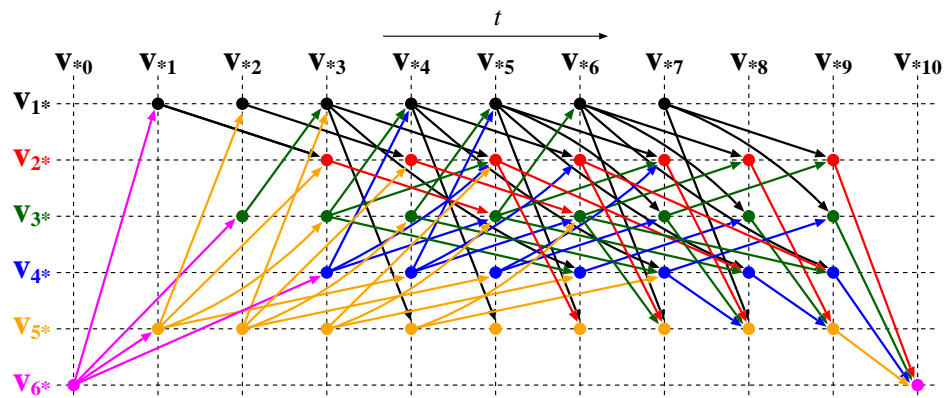


Figure 3: An Example of the Network \hat{G}_S ($p_1 = 1, p_2 = 2, p_3 = 2, p_4 = 3, p_5 = 1$, a precedence constraint: $1 \rightarrow 2$)

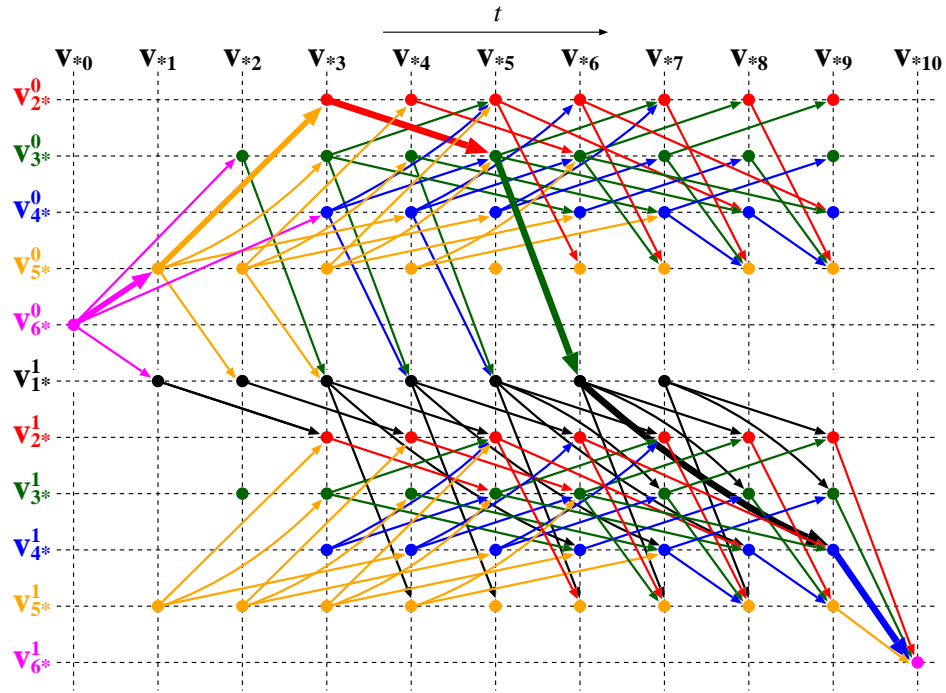


Figure 4: An Example of the Network G_S^m ($p_1 = 1, p_2 = 2, p_3 = 2, p_4 = 3, p_5 = 1$, a precedence constraint: $1 \rightarrow 2, \mathcal{M} = \{1\}, m = 1$)

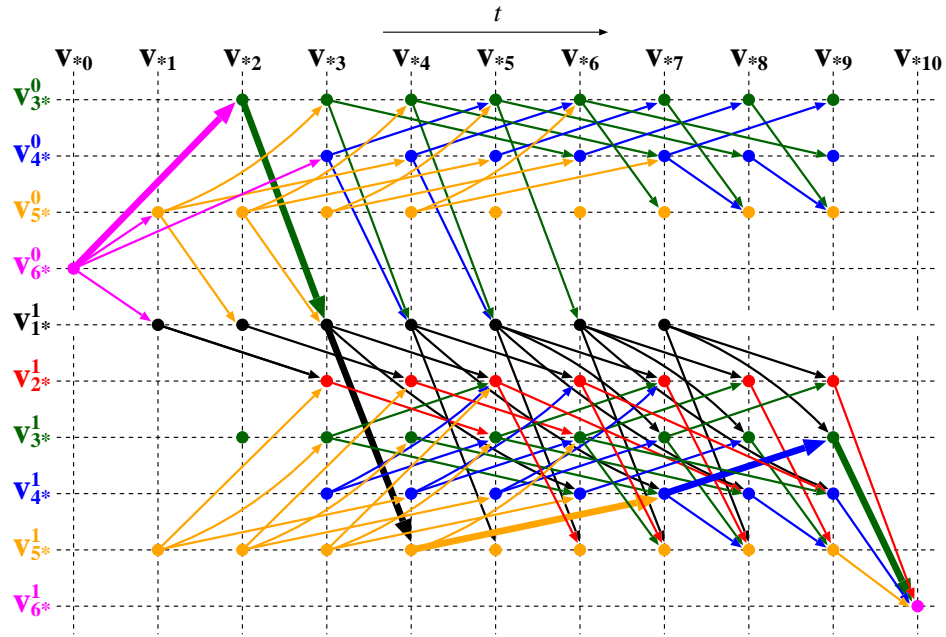


Figure 5: An Example of the Network \hat{G}_S^m ($p_1 = 1, p_2 = 2, p_3 = 2, p_4 = 3, p_5 = 1$, a precedence constraint: $1 \rightarrow 2, \mathcal{M} = \{1\}, m = 1$)